Manual

de utilização de

4GL Interactive Debugger

1.Introdução

No desenvolvimento de programas utilizando linguagem procedimentais tradicionais, um dos maiores custos do desenvolvimento centra-se na correcção dos erros dos programas desenvolvidos e (ou) a desenvolver.

Existem básicamente dois tipos de erros: Erros de ***sintaxe ou semântica*** em que os programas estão escritos de uma forma errada ou têm sequência de instruções incorrectas, e os erros de ***funcionamento*** *(run time)* que não inibem a compilação dos programas, mas que provocam um funcionamento errado destes. A estes últimos é costume chamar ***bug*** .

Existem utilitários que permitem a monitorização do funcionamento dos programas, permitindo um conjunto de acções sobre a execução dos programas. A este utilitário chama-se debbuger (também chamado depurador em brasileiro).

A INFORMIX, para a sua principal linguagem procedimental (Informix 4GL) desenvolveu um debugger ao qual chamou ***informix interactive debugger***. Este programa funciona apenas com a versão RDS do informix 4GL e trabalha em terminais em modo caracter, e mais recentemente também em windows.

2.Execução do Interactive Debbuger

Tal como em outros produtos Informix, é possivel executar o utilitário através de menus ou de linha de comando do Unix ou Dos.

2.1.Opção "debug"do r4gl

Uma das formas de execução através da opção "Debug" do menu Programa ou Debug do Informix 4GL RDS (comando r4gl).

Neste caso o programa a ser corrigido é o que foi escolhido por um écran de escolha, muito usual no Informix.

2.2.Linha de comando

Para fazer debug (depurar) um programa através da linha de comando do shell, deverá executar o comando *fgldb*, cujo sintaxe é a seguinte:

fgldb { -v [-I pathname[,pathname,...]] -f initfile name4gi}

O programa 4gi será o executável a ser analisado. As outras opções têm o seguinte significado:

- V - Serve para mostrar apenas a versão do debbuger que estamos a utilizar.

- I pathname - Serve para dizer ao debbuger onde deverá (para além da directória corrente) ir procurar os programas fonte do 4GL.

- F nome_ficheiro - Serve para dizer ao debbuger que pretendemos utilizar um ficheiro de configuração alternativo ao standard do informix.

3.Janelas do ID

O interactive debugger, como o seu nome indica é interactivo, isto é, permite uma interacção com o utilizador.

Neste produto a interacção é feita através de janelas com posicionamentos directos do cursor, tentando utilizar todas as características dos terminais de caracteres.

Tendo em atenção estas limitações de ambiente, a tarefa do debbuging coloca-nos o seguinte problema:

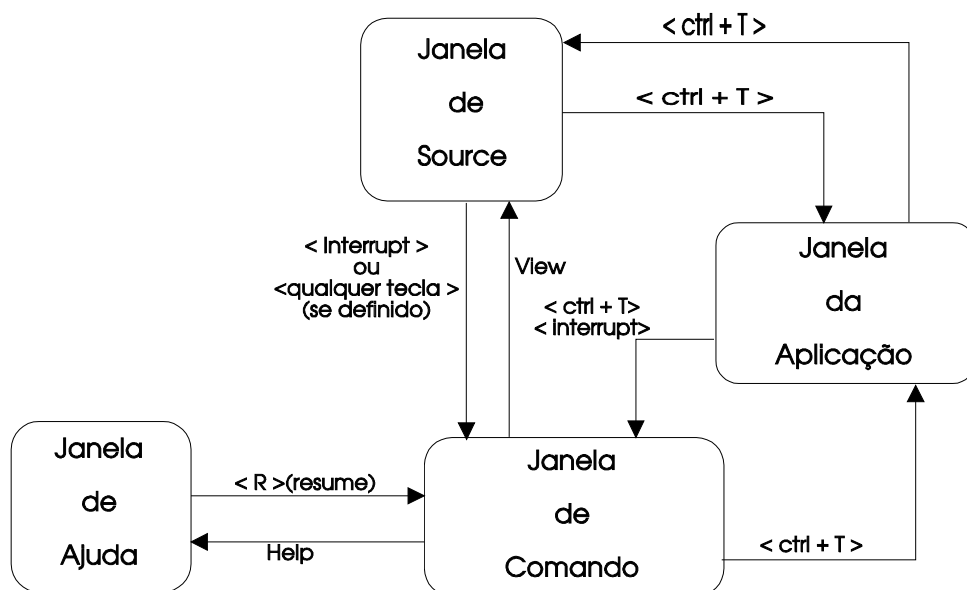
- A nossa aplicação / programa a examinar trabalha num écran e recebe comandos de um teclado.

- O utilitário de debbuging tem de nos mostrar resultados e receber comandos de um teclado.

Para resolver estes problemas virtualizaram-se (simularam-se) janelas nos terminais permitindo uma fácil comutação e configuração consoante o programador o pretenda.

Quando executado, o ID apresenta a janela de debug com as respectivas duas sub-janelas. É apresentado na janela de comandos a prompt que indica que se está á espera de comandos.

As janelas existentes são as seguintes: Janela de debug; janela de help; janela de aplicação.

Para comutar entre as diferentes janelas, existem comandos especificos no ID.
A figura que de seguida se apresenta tenta, de uma forma gráfica identificar as janelas e respectivos comandos de comutação:

3.1.Janela de debugg

Esta janela, por sua vez encontra-se dividida em duas sub-janelas: ***janela do source*** de programa e ***janela de comandos***.

> **Janela de source**: Nesta sub-janela é identificado o módulo que estamos a visualizar e o respectivo programa. Esta janela está na parte superior do écran.

> **Janela de comandos**: Nesta janela são visualizados os comandos digitados e o seu resultado. Está localizado na parte inferior do écran.

3.2.Janela da aplicação

Esta janela, do tamanho do écran tem o resultado da execução do programa. Para comutar entre a janela de aplicação e janela de debbug utiliza-se a tecla CTRL+T.
Esta janela poderá ser fácilmente colocada noutro terminal, para visualização simultânea da janela de debug e da aplicação. Para tal utiliza-se o comando *Application Device*.

3.3.Janela de Help

Nesta janela serão mostrados écrans de ajuda, com texto explicativo do comando acerca do qual se tem dúvidas.

Para aceder a esta janela, digita-se o comando *help*, cuja sintaxe é a seguinte:

HELP [comand |ALL]

Desta forma, é possivel aceder a uma ajuda referente apenas a um comando (digitando o nome específico do comando), a todas (opção ALL) ou aparecendo um menu que permite escolher o comando de que se pretende obter ajuda.

3.4.Janela de Comando

Nesta janela poderão ser dados (ao debugger) comandos para análise dos programas.
Tal como no shell, o comando após ser digitado termina com **<enter>.**

O comando *grow* permite controlar o tamanho das janelas debugg. A sua sintaxe é a seguinte:

GROW [SOURCE | COMMAND] [-] NÚMERO

O número passado como argumento é o nº de linhas que a janela de source vai mudar. Se pretender a janela de source e aumentar a do comando deverá executar o *grow* com um número negativo.

Quando executa o comando grow sem opções o debugger escreve que se está a mudar a janela de source (consequentemente assume a opção SOURCE).

Se em vez de utilizar números negativos pretender fazer crescer a janela de comando deve especificara opção COMMAND.

Tendo a seguinte janela, usual quando se executao ID, vamos agora diminuir a janela de source:

```
+-----------------------------------------------------------------------------+
¦    1   main                                                                  ¦
¦    2           display "HELLO WORLD"                                         ¦
¦    3   end main                                                              ¦
¦(exp.4gl:main)                                                                ¦
+-----------------------------------------------------------------------------+
+-----------------------------------------------------------------------------+
¦$grow -6                                                                      ¦
¦$                                                                             ¦
¦                                                                              ¦
+-----------------------------------------------------------------------------+
```

Obtendo-se:

```
+-----------------------------------------------------------------------------+
¦    1   main                                                                  ¦
¦    2           display "HELLO WORLD"                                         ¦
¦(exp.4gl:main)                                                                ¦
+-----------------------------------------------------------------------------+
```

```
+---------------------------------------------------------------------------+
¦$grow -7                                                                   ¦
¦$list display                                                              ¦
¦TERMINAL DISPLAY STATE                                                     ¦
¦autotoggle          on                                                     ¦
¦displaystops        on                                                     ¦
¦sourcetrace         off                                                    ¦
¦exitsource          on                                                     ¦
¦printdelay          off                                                    ¦
¦timedelay source    1                                                      ¦
¦timedelay command   0                                                      ¦
¦source lines        2                                                      ¦
¦command lines       17                                                     ¦
¦$                                                                          ¦
¦                                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
+---------------------------------------------------------------------------+
```

De notar que este comando também poderia ser digitado da seguinte forma:

grow command 7

Análogamente também se poderia diminuir a janela de comando com um dos seguintes comandos:

grow 7

grow source 7

grow comand -7

Obtendo-se:

```
+---------------------------------------------------------------------------+
¦    1   main                                                               ¦
¦    2           display "HELLO WORLD"                                      ¦
¦    3   end main                                                           ¦
¦                                                                           ¦
¦(exp.4gl:main)                                                             ¦
+---------------------------------------------------------------------------+
+---------------------------------------------------------------------------+
¦$                                                                          ¦
¦$grow 7                                                                    ¦
¦$                                                                          ¦
+---------------------------------------------------------------------------+
```

3.5.Janela do Programa Fonte

Na janela de source, quando se entra no debugger aparece o módulo onde está a função "main", do nosso programa.

Para passar da janela de comando para a janela de source, utiliza-se o *comando view.*

Na janela de source poderá dar vários comandos, por forma a permitir a localização e posicionamneto do código fonte pretendido:

<control+J> - Move o cursor uma linha para baixo
<control+K> - Move o cursor uma linha para cima
<control+B> - Move o cursor uma janela para cima
<control+F> - Move o cursor uma janela para baixo
<control+U> - Move o cursor meia janela para cima
<control+D >- Move o cursor meia janela para baixo

n [tecla_controle] - sendo n um nº executa a acção definitiva pela tecla de controle n vezes

n [return] - move-se para a linha n do programa

$ - Vai para o fim do ficheiro.

/ **string_de_busca** - Procura uma frase de acordo com a string de busca definida

[return] - move-se para a próxima ocorrência da última busca introduzida.

3.6.Comando VIEW

Com o comando *view*, é possível passar da janela de comando para a janela do programa fonte.

A sua sintaxe é a seguinte:

VIEW [módulo | função]

Se não fôr atribuído nenhum nome do módulo ou da função, a acção irá passar para a janela de source, do programa corrente.

Se fôr descriminado um módulo ou função, e caso este não se refira ao source corrente, aparecerá na janela de source o respectivo programa fonte.

No exemplo que se segue, o nosso programa encontra-se dividido em dois módulos (exp.4gl e exp1.4gl), estando o main no módulo exp.4gl e a única função (func1) no módulo exp1.4gl.

Quando executamos o ID, é-nos apresentada a janela de source com o módulo onde se encontra a função main:

```
+--------------------------------------------------------------------------+
¦     1   main                                                             ¦
¦     2     define                                                         ¦
¦     3       ans smallint                                                 ¦
¦     4                                                                    ¦
¦     5     let ans=func1()                                                ¦
¦     6                                                                    ¦
¦     7   end main                                                         ¦
¦                                                                          ¦
¦                                                                          ¦
¦(exp.4gl:main)                                                            ¦
+--------------------------------------------------------------------------+
+--------------------------------------------------------------------------+
¦$view exp1                                                                ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
+--------------------------------------------------------------------------+
```

De notar que, no comando *view* não foi apresentada a extensão, (4gl) do ficheiro o que não poderá fazer pois o ID irá dar um erro. O resultado será o seguinte:

```
+--------------------------------------------------------------------------+
¦     1   function func1()                                                 ¦
¦     2   {}                                                               ¦
¦     3                                                                    ¦
¦     4     display "HELLO WORLD"                                          ¦
¦     5     return 1                                                       ¦
¦     6                                                                    ¦
¦     7   end function                                                     ¦
¦                                                                          ¦
¦                                                                          ¦
¦(exp1.4gl:func1)                                                          ¦
+--------------------------------------------------------------------------+
+--------------------------------------------------------------------------+
¦$view exp1                                                                ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
+--------------------------------------------------------------------------+
```

Para visualizarmos a função *func1* poderíamos, opcionalmente ter digitado o comando *view* com a função como parâmetro. Assim poderia ter digitado os seguintes comandos:

view func1 - O ID apresenta a função func1 na janela de source.

view main - O ID apresenta a função main na janela de source

3.7. USE

Alteração de caminho de procura.

O comando *use* serve para especificar a(s) directoria(s) de busca dos programas fonte. Se nada em contrário for especificado (com o parâmetro -I ou variável DBSRC), o único sítio onde o debugger irá tentar buscar os programas fonte será à directoria corrente.

Com o comando *use* pode adicionar novos caminhos. A sua sintaxe é a seguinte:

USE [[=] pathname [,...]]

Se digitar o comando *use* sem qualquer argumento, será apresentado no écran o caminho de busca utilizado até aqui. Se utilizar o caracter "=" (igual), está a dizer ao debugger para esquecer o caminho anterior anterior e utilizar apenas o especificado.

Se digitar o comando sem igual ("=") então estará a adicionar ao caminho antigo, um caminho adicional.

Exemplo 1:

$ fgldb exp

```
+--------------------------------------------------------------------------------+
¦    1   main                                                                    ¦
¦    2           display "HELLO WORLD"                                           ¦
¦    3   end main                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦(exp.4gl:main)                                                                  ¦
+--------------------------------------------------------------------------------+
+--------------------------------------------------------------------------------+
¦$use                                                                            ¦
¦Current search path: , .                                                        ¦
¦$use /tmp                                                                       ¦
¦Current search path: /tmp, , .                                                  ¦
¦$                                                                               ¦
¦                                                                                ¦
```

De acordo com o exemplo apresentado, quando se executar o ID, o nosso caminho de procura dos source(s) é a directoria corrente (.), conforme especificado (current search path).

Uma vez adicionada uma nova directoria (/tmp) são apresentadas no écran duas directorias (de busca dos sources a analisar) (/tmp, .).

Exemplo 2:

```
+--------------------------------------------------------------------------------+
¦    1   main                                                                    ¦
¦    2           display "HELLO WORLD"                                           ¦
¦    3   end main                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦(exp.4gl:main)                                                                  ¦
+--------------------------------------------------------------------------------+
+--------------------------------------------------------------------------------+
¦$use                                                                            ¦
¦Current search path: , .                                                        ¦
¦$use:/tmp                                                                       ¦
¦Current search path:  /tmp, , .                                                 ¦
¦$                                                                               ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
¦                                                                                ¦
```

```
+---------------------------------------------------------------------------+
```

Neste exemplo, antes de executarmos o ID movemos o nosso source para a directoria anterior.

Uma vez executado, o ID informa-nos imediatamente que não consegue encontrar o programa.

Quando mudamos o *search path* também para a directoria corrente, o source já é encontrado.

4.Facilidades Operativas

4.1.Aliases

Por forma a aumentar a produtividade, foram criadas algumas facilidades de operação,. De entre outras destaca-se o *alias.*

Um *alias* é, tal como o seu nome indica uma forma de dar um nome alternativo a um comando. Desta forma, um comando muito comprido, que é consequentemente muito moroso na sua escrita pode ser redefinido com um nome pequeno, de fácil e rápida digitação.

Exemplo:

```
        alias prc = print pr_automóvel.caixa [1,20]

        prc
                pr_automóvel.caixa [1,20] = "fechada"
```

Conforme pode ser verificada, depois de feito o *alias* bastou digitar o *alias* para executar o comando pretendido.

Esta facilidade, se for aliada à escrita e leitura de comandos em ficheiros, irá permitir um grande aumento de produtividade na análise a programas.

A sintaxe genérica deste comando é descrita na seguinte figura:

```
ALIAS
{
        *
        |
        nome = comando
        |
        nome = { comando [; comando ...]}
}
```

Conforme pode verificar poderá utilizar três tipos de argumentos alternativos. O primeiro destes (*) serve para que o programa mostre (lista) os *aliases* existentes.

Existe diferença entre a sintaxe de utilização de um ou mais comandos pois com mais do que um comando deverá colocar os comandos entre chavetas ({}) e separar por ";".

Se o pretender pode definir um alias constituído por *aliases*, no entanto não poderá meter mais de 5 níveis de aliases misturados.

```
+-------------------------------------------------------------------------+
¦    1   main                                                              ¦
¦    2           display "HELLO WORLD"                                     ¦
¦    3   end main                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦(exp.4gl:main)                                                            ¦
+-------------------------------------------------------------------------+
+-------------------------------------------------------------------------+
¦$alias ve_tudo = dump all                                                 ¦
¦$alias liga_breaks = {enable 1 ; enable 2}                                ¦
¦$alias desliga_breaks = {disable 1 ; disable 2}                           ¦

¦$alias                                                                    ¦
¦-16301: A syntax error has occurred.                                      ¦
¦$                                                                         ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
+-------------------------------------------------------------------------+
```

Neste exemplo foram definidos vários aliases, tanto para apenas um comando como para um conjunto de comandos. Para ver os aliases definidos, bastou fazer ***alias.***

4.2.Direccionamento do Output (>>)

Alguns comandos do ID, dão por vezes resultados que nem sequer cabem no écran, tornando-se impossível utilizar o seu resultado para o que quer que seja.

Foram, no entanto dotados de uma capacidade de escrever o resultado num ficheiro do sistema operativo, que posteriormente se poderá examinar.

```
+-------------------------------------------------------------------------+
¦    1   main                                                              ¦
¦    2           display "HELLO WORLD"                                     ¦
¦    3   end main                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦                                                                          ¦
¦(exp.4gl:main)                                                            ¦
+-------------------------------------------------------------------------+
+-------------------------------------------------------------------------+
¦$dump all >>variaveis                                                     ¦
¦$                                                                         ¦
¦                                                                          ¦
```

O comando ***dup all*** mostra o conteúdo de todas as variáveis. Numa aplicação de médio porte é muito natural que as variáveis não apareçam no écran, tornando-se pois difícil de examinar.

Para facilitar a busca e posterior análise enviou-se o output para dentro de um ficheiro, que posteriormente poderá ser examinado com um qualquer editor de texto (por ex: o ***vi***)

4.3.Application device

O exame a determinados programas pode tornar-se confuso, principalmente se fôr necessário o exame dos écrans.

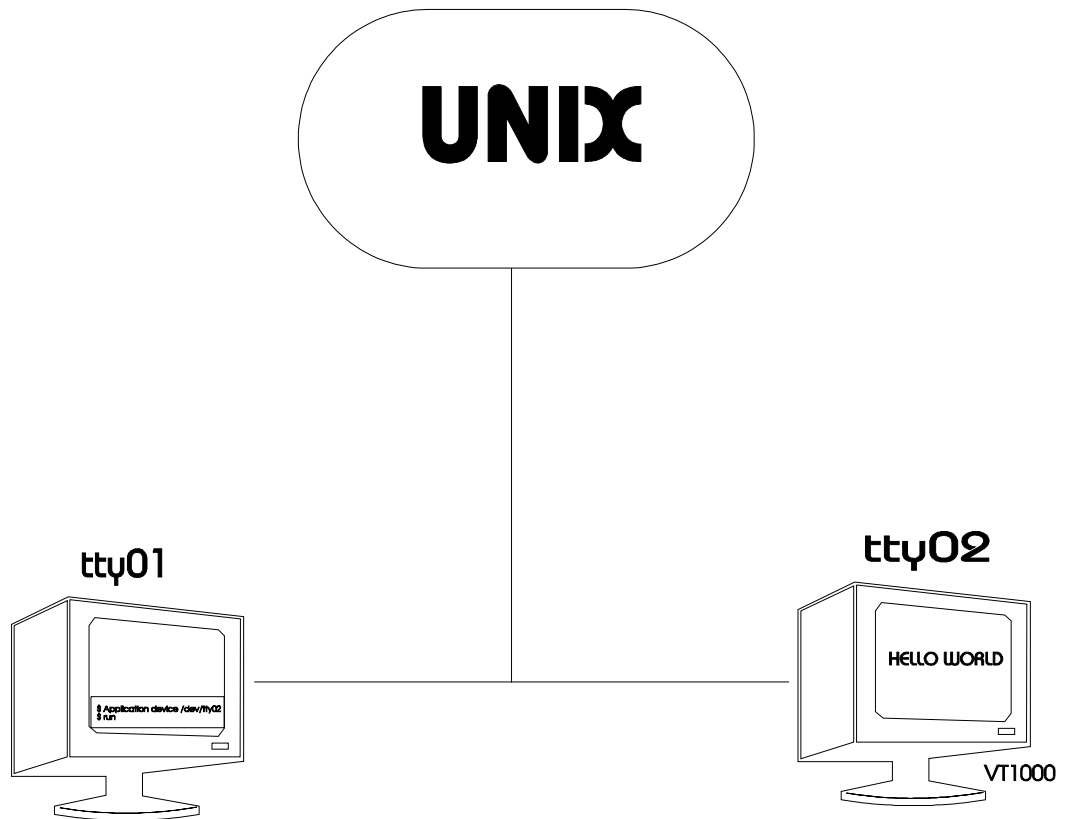
O informix ID foi dotado de uma instrução que permite utilizar o écran do seu terminal para as janelas de debugg e comando, e utilizar outro écran para o resultado da aplicação.

A sua sintaxe é a seguinte:

APPLICATION [DEVICE] nome - do - device

O nome-do-device deverá incluir o pathname completa do device (p/ex /dev/tty01).
Para tornar mais rápido poderá não digitar a palavra DEVICE.
Para utilizar em sucesso esta capacidade deverá obedecer ás seguintes regras:

- O tipo de terminal deverá ser igual (receber as mesmas sequências de comando)
- Deverá ter entrado no sistema operativo com o mesmo programa, nos dois terminais.

Supondo que tinha dois terminais vt100 em cima da secretária e pretendia utilizar um para visualizar o resultado do programa e a outra para dar comandos e monitorizar a sua execução.

Supondo que se encontra a trabalhar no terminal tty01 e pretendia enviar o resultado para o tty02, então deveria digitar o seguinte comando, na linha de comando do ID :

```
+------------------------------------------------------------------------------+
|    1   main                                                                  |
|    2           display "HELLO WORLD"                                         |
|    3   end main                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|(exp.4gl:main)                                                                |
+------------------------------------------------------------------------------+
+------------------------------------------------------------------------------+
|$Application device /dev/tty02                                                |
|                                                                              |
|                                                                              |
|                                                                              |
+------------------------------------------------------------------------------+
```

4.4.TURN

Serve para ligar e desligar os parâmetros do funcionamento do écran.

Para ligar e (ou) desligar alguns dos parâmetros de funcionamento do écran utiliza-se a instrução *turn.* Os argumentos desta instrução dividem-se em dois tipos distintos:

- Acção a efectuar (ligar ou desligar)
- Parâmetro a alterar

Se não fôr especificada a acção, o parâmetro será ligado (ON).

A sintaxe genérica deste comando é:

```
TURN [ON | OFF]

{
    AUTOTOGGLE
    |
    DISPLAY STOPS
    |
    EXIT SOURCE
    |
    PRINTDELAY
    |
    SOURCETRACE
}...
```

Conforme se pode verificar, desde que se pretenda efectuara mesma opção (ligar ou) é possivel efectuá-la sobre mais do que um parâmetro de cada vez.

Se não fôr executada nenhuma operação de alteração sobre os parâmetros serão assumidas as seguintes configurações:

AUTOTOGGLE - ON

DISPLAYSTOPS - ON

EXITSOURCE - ON

PRINTDELAY - OFF

SOURCETRACE - OFF

4.5.WHERE

Apresenta a pilha de funções com respectivos argumentos de execução.

Quando se analisa um programa, muitas vezes necessitamos saber quais as funções activas em determinado momento e quais os argumentos com que foram executados.
Com a instrução *where* é apresentado esse stock de execução. Utilizando o redirecionamento poderá enviar para dentro de um ficheiro.

WHERE [>> nome-ficheiro]

Esta instrução é ainda válida se o programa abortar com um erro fatal. Esta capacidade é muito importante uma vez que permite saber as funções activas (caminho de execução) na altura em que o programa deu erro .
Não pode utilizar esta instrução se não existir nenhum programa ou função activa.

5.Executar comandos do S.O

Tal como na maioria dos utilitários informix, também aqui é possivel executar o shell ou outros comandos. Esta acção será feita utilizando o caracter "!", escrevendo de seguida o comando pretendido.

Exemplo:

Se pretender executar o shell, por forma a ficar c/ a prompt do shell sem sair da sua sessão de debug deverá fazer:

!sh
$ls
$control + D

```
+----------------------------------------------------------------------------+
¦    11 MAIN                                                                 ¦
¦    12                                                                      ¦
¦    13    DEFER INTERRUPT                                                   ¦
¦    14                                                                      ¦
¦    15    OPEN FORM cust_form FROM "customer"                               ¦
¦    16                                                                      ¦
¦    17    DISPLAY FORM cust_form                                            ¦
¦    18                                                                      ¦
¦    19    LET chosen = FALSE                                                ¦
¦(customer.4gl:main)                                                         ¦
+----------------------------------------------------------------------------+
+----------------------------------------------------------------------------+
¦$!ls                                                                        ¦
¦                                                                            ¦
¦                                                                            ¦
¦                                                                            ¦
¦                                                                            ¦
¦                                                                            ¦
¦                                                                            ¦
¦                                                                            ¦
¦                                                                            ¦
+----------------------------------------------------------------------------+
customer.frm
customer.per
customer.unl
customer.4gi
customer.4gl
customer.4go
my.4db
my.4gl
my.4go
stores.dbs
syspgm4gl.dbs
tmp.4db

 Press RETURN to continue
```

6.Execução de um programa ou função

O comando no ID que permite a execução do programa é o comando *run.* Se o pretender poderá também executar apenas uma função específica. Para tal utiliza-se o comando *call.*

6.1. CALL

Execução de função isolada

Muito frequentemente a análise deverá ser apenas a uma função específica. Não é prática a execução de todo o programa metendo pontos de paragem para chegar à zona onde é executada a função a examinar.

Para executar apenas a função utiliza-se o comando *call*. A sua sintaxe é idêntica à da instrução call na sua linguagem 4GL:

CALL nome função(parâmetro [,...])

Quando a execução da função terminar, se esta retomar valores, o utilizador será informado dos valores retornados na janela de comando.

```
+-----------------------------------------------------------------------------+
¦    1   main                                                                 ¦
¦    2           define ans smallint                                          ¦
¦    3   call func1 () returning ans                                          ¦
¦    4   end main                                                             ¦
¦    5                                                                        ¦
¦    6   function func1()                                                     ¦
¦    7   return 1                                                             ¦
¦    8   end function                                                         ¦
¦                                                                             ¦
¦(exp.4gl:func1)                                                              ¦
+-----------------------------------------------------------------------------+
+-----------------------------------------------------------------------------+
¦$call func1()                                                                ¦
¦Return  (1) from func1 at line 7                                             ¦
¦Program exited in func1 at line 7 in module "exp.4gl"                        ¦
¦$                                                                            ¦
¦                                                                             ¦
¦                                                                             ¦
¦                                                                             ¦
```

Se pretender executar a função várias vezes seguidas e a função inicializar janelas ou instruções sem as funções, deverá utilizar a instrução ***cleanup*** para efectuar uma limpeza.

6.2. RUN

Execução do programa

O comando *run* inicia a execução do programa que se está a examinar. Se o programa interactuar com o écran e a opção *autotoggle* se encontrar ligada, o ID irá comutar o écran para a janela do programa.

Se o programa tiver parâmetros, a sua execução faz-se enviando os respectivos parâmetros a seguir à instrução *run*.

A sintaxe do comando *run* é a seguinte:

RUN [parâmetro] [...]

```
+---------------------------------------------------------------------------+
|    1   main                                                               |
|    2   define frase char (30)                                             |
|    3   let frase = arg_val(frase)                                         |
|    4   display frase                                                      |
|    5   end main                                                           |
|    6                                                                      |
|                                                                           |
|                                                                           |
|                                                                           |
|(exp.4gl:main)                                                             |
+---------------------------------------------------------------------------+
+---------------------------------------------------------------------------+
|$run "HELLO WORLD"                                                         |
|                                                                           |
```

Digitando control + T fica-se posicionado na janela da aplicação, onde irá aparecer:

HELLO WORLD

6.3.CLEANUP

Limpeza de ambiente

Se estiver a meio de uma execução (do programa ou função), poderão existir janelas ou forms já abertas, e eventualmente variáveis inicializadas com determinados valores.

Para limpar este ambiente e repôr essas condições iniciais de execução do programa utiliza-se a instrução *cleanup* (acção de limpar).

A já nossa conhecida sintaxe genérica é representada da seguinte forma:

CLEANUP [ALL]

A execução do *cleanup*, só por si limpa todas as janelas, forms, e variáveis. A base de dados continua aberta.

Se especificar a opção ALL a base de dados corrente será também encerrada.

Exemplo:

```
$ cleanup
$ cleanup all
```

7.Modos de execução

No ID, ao contrário da execução normal, um programa pode ser executado de várias formas: execução continua; execução instrução a instrução; execução contínua com visualização da instrução corrente.

7.1.Execução passo a passo

Pode executar o programa ou função, instrução a instrução apenas a partir de uma interrupção no programa. Se desejar executar passo a passo a partir do ínicio deverá meter um ponto de paragem no ínicio da função main[1]:

```
$BREAK MAIN
```

Para executar passo a passo o programa, deverá digitar o comando *step*, ou o seu *alias* (f2).
Na janela do source, a instrução que será executada de seguida é apresentada assinalada com reverse.

$ fgldb exp

```
+--------------------------------------------------------------------------
--+
¦    1   main
¦
¦    2   display "HELLO WORLD"
¦
¦    3   display "HERE I GO"
¦
¦    4   end main
¦
¦    5
¦
¦
¦
¦
¦
¦(exp.4gl:main)
¦
+--------------------------------------------------------------------------
--+
+--------------------------------------------------------------------------
--+
¦$break 2
¦
¦(1)  break main:2 [exp.4gl]
¦
¦          scope function: main
¦
¦$run
¦
¦Stopped in main at line 2 in module "exp.4gl"
¦
¦$
¦
¦
```

[1]Ver pontos de paragem ou breakpoints

Conforme se pode ver, o programa parou a execução na linha 2 do programa fonte, deixando-a marcada com reverse. Este facto significa que a próxima linha a ser executada será esta.

```
+-------------------------------------------------------------------------
--+
¦    1   main
¦
¦    2   display "HELLO WORLD"
¦
¦    3   display "HERE I GO"
¦
¦    4   end main
¦
¦    5
¦
¦
¦
¦
¦(exp.4gl:main)
¦
+-------------------------------------------------------------------------
--+
+-------------------------------------------------------------------------
--+
¦$step
¦
¦-16338: Cannot continue execution.
¦
¦$
¦
¦
¦
+-------------------------------------------------------------------------
--+
```

Quando foi digitado o comando step, a instrução "display "HELLO WORLD" " foi executada, e o display seguinte foi marcado para execução.

7.2.Execução continua

Neste modo o programa, ou função é executada segundo a ordem definida, não mostrando onde se encontra, apenas executando as instruções.

Se não meteu pontos de paragem, tem a opção *autototggle* ligada e não carregar em interrupt a execução do programa faz-se tal como se estivesse a executar o programa normalmente.

Para parar o programa em determinada altura deverá carregar em delete. Quando o fizer, o comando será imediatamente passado ao debugger e apresentada a janela de comando.

Para executar neste modo utilizam-se os comandos *run* ou *call*.

7.3.Timedelay

Alterar tempos de paragem entre execuções.

Se continuarmos a executar o programa com tracejamento de source continuamente (sem step), poderemos querer definir um maior ou menos tempo de paragem no levar para cada paragem. Esta configuração pode também ser definida para o envio das linhas resultantes de um comando para a janela do comando (por exemplo um print).
A sintaxe è a seguinte:

TIMEDELAY [SOURCE | COMMAND] n

Se nenhum argumento (source ou command) fôr especificado, ou *timedelay* irá incidir sobre a janela de source.

Quando especificar o parâmetro comand, esta terá incidência sobre o resultado dos comandos.

O número é colocado no fim das instruções, sendo obrigatória define ao número de segundos que o programa mete de atraso. Se este comando não fôr especificado estes parâmetros estão definidos a um . Pode definir o parâmetro a 0 ou a qualquer número positivo.

Para verificar o valor deste comando utiliza-se o comando *list*.

Exemplo:

$ Timedelay source 2 - Espera dois segundos entre cada instrução executada.

$ Timedelay source 0 - O programa não pára tempo nenhum entre cada instrução (do programa) executada.

$ Timedelay comand 2 - Entre cada linha mostrada na janela de comando será feita uma paragem de 2 segundos.

No seguinte exemplo assume-se a diminuição da janela de source em 7 linhas:

```
+---------------------------------------------------------------------------
--+
¦    1   main
¦
¦    2     define
¦
¦(exp.4gl:main)
¦
+---------------------------------------------------------------------------
--+
+---------------------------------------------------------------------------
--+
¦$grow -7
¦
¦$timedelay command 2
¦
¦$list display
¦
¦TERMINAL DISPLAY STATE
¦
¦autotoggle          on
¦
¦displaystops        on
¦
¦sourcetrace         off
¦
```

```
¦exitsource          on
¦printdelay          off
¦timedelay source    1
¦timedelay command   2
¦source lines        2
¦command lines       17
¦$
¦
¦
¦
¦
+-----------------------------------------------------------------------------
--+
```

Tendo sido alterado o atraso no mostra na janela de comando, o aparecimento do resultado do ***list*** é considerávelmente mais lento pois um atrazo de 2 segundos aparece entre cada linha enviada para o écran.

7.4.Step

Executa um ou mais comandos de 4gl

Quando se está a trabalhar com execução passo a passo, é necessário um comando para permitir a execução de mais um passo. Esse comando é o *step*, e para além de permitir a execução de mais um passo permite ainda a execução de um passo e/ou assumir uma função como um único passo e ainda dizer para não ligar aos *breakpoints* definidos por onde passe.

A sua sintaxe é:

STEP [n] [INTO] [NOBREAK]

Conforme se pode verificar, o nome que se especifica a seguir à instrução *step* opcionalmente define o número de instruções a executar para este passo definido.

Se o parâmetro *into* fôr especificado, uma função será assumida como um passo único, caso contrário o debugger entrará dentro da função executando por sua vez cada passo deste.

Nobreak utiliza-se para fazer com que o debugger não páre nos breakpoints definidos por onde passe.

```
+----------------------------------------------------------------------------+
¦    1   main                                                                ¦
¦    2           display "HELLO WORLD"                                       ¦
¦    3           display "HERE I AM"                                         ¦
¦    4           display "VENI VIDI VICI"                                    ¦
¦    5   end main                                                            ¦
¦    6                                                                       ¦
¦                                                                            ¦
```

```
¦                                                                          ¦
¦(exp.4gl:main)                                                            ¦
+--------------------------------------------------------------------------+
+--------------------------------------------------------------------------+
¦-16352: File [exp.4gl] has been modified. (.4gl is newer than .4go)       ¦
¦$break main                                                               ¦
¦(1)  break in function main [exp.4gl]                                     ¦
¦          scope function: main                                            ¦
¦$run                                                                      ¦
¦Stopped in main at line 2 in module "exp.4gl"                             ¦
¦$step 2                                                                   ¦
¦Stopped in main at line 4 in module "exp.4gl"                             ¦
¦$                                                                         ¦
¦                                                                          ¦
+--------------------------------------------------------------------------+
```

Assumindo que os 2 primeiros comandos já teriam sido executados, depois do ***run,*** o programa ficaria parado antes de executar a instrução dsiplay "HELLO WORLD".

Como se deu a ordem de executar um passso de duas instruções (step 2), o programa ficou parado na instrução da linha 4.

7.5.Continue

Continua a execução depois de uma paragem e permite enviar sinais à aplicação.Quando paramos a execução do progrma (através de carregar em *del, ou* através de um *breakpoint)* podemos continuá-la em execução contínua através do comando *continue.*

A sua sintaxe genérica é:

CONTINUE [INTERRUPT | QUIT]

As duas opções (CONTINUE E QUIT) servem para permitir o envio dos sinais *Interrupt* e *Quit* à aplicação.

Esta capacidade torna-se necessário pois o debugger apanha os *sinais* devolvendo o comando à janela de comando.

No exemplo que se segue, o programa foi configurado, por forma a utilizar o interrupt para abortar forms e outras acções, activando a variável int_flag.

Por forma a activar efectivamente esta variável, temos que executar a instrução *continue interrupt:*

```
+--------------------------------------------------------------------------+
¦    3   main                                                              ¦
¦    4      define                                                         ¦
¦    5                   pr_discos record like discos.*                    ¦
¦    6                                                                     ¦
¦    7      defer interrupt                                                ¦
¦    8                                                                     ¦
¦    9           open window w_discos at 4,6 with form "disco"             ¦
¦   10                                                                     ¦
¦   11           input by name pr_discos.*                                 ¦
¦   12                                                                     ¦
¦   13           if int_flag then                                          ¦
¦   14                   error "Inserçao abortada"                         ¦
¦   15           end if                                                    ¦
¦   16   end main                                                          ¦
¦(discos.4gl:main)                                                         ¦
+--------------------------------------------------------------------------+
+--------------------------------------------------------------------------+
¦$run                                                                      ¦
¦Stopped in main at line 7 in module "discos.4gl"                          ¦
¦$continue interrupt                                                       ¦
¦Program aborted in main at line 7 in module "discos.4gl"                  ¦
¦$                                                                         ¦
+--------------------------------------------------------------------------+
```

Na altura em que foi dado a instrução *continue interrupt,* o programa continuou a sua execução, tendo a variável int_flag activada, o que faz com que fosse dada a mensagem no écran:

INSERÇÃO ABORTADA
em reverse

8.Manuseamento de variáveis

Para analisar um programa, torna-se essencial a visualização do conteúdo das variáveis.
O ID tem várias formas de efectuar esta tarefa, para diferentes casos.

8.1.Print

O comando *print* envia para o écran (ou coloca num ficheiro) o resultado de uma expressão.
A sua sintaxe genérica é:

PRINT expressão [>> nome-ficheiro]

Conforme se pode verificar, o redireccionamento é opcional, caso não o indique o resultado será enviado para a janela de controle.
A expressão poderá ser a definição de uma das seguintes entidades:

- **variável simples**
- **um record**
- **uma função de data ou data/tempo**
- **um array**
- **uma expressão aritmética**
- **uma string entre aspas**

Só poderá inspeccionar variáveis globais, ou variáveis locais à função que está activa no momento em que faz o *print*.
A visualização do conteúdo da variável só poderá ser feita depois de ter sido feito um *call* ou *run*.

```
+------------------------------------------------------------------------------+
|     2   main                                                                 |
|     3           define pr_discos record like discos.*                        |
|     4                                                                        |
|     5           select * into pr_discos.*                                    |
|     6           from discos                                                  |
|     7           where discos.numero = 1                                      |
|     8                                                                        |
|     9           display "DISCO 1:", pr_discos.nome                           |
|    10   end main                                                             |
|(discos.4gl:main)                                                             |
+------------------------------------------------------------------------------+
+------------------------------------------------------------------------------+
|$break 9                                                                      |
|(1)  break main:9 [discos.4gl]                                                |
|          scope function: main                                                |
|$run                                                                          |
|Stopped in main at line 9 in module "discos.4gl"                              |
|$                                                                             |
|                                                                              |
```

```
|                                                                          |
|                                                                          |
+--------------------------------------------------------------------------+
```

```
+--------------------------------------------------------------------------+
|    2   main                                                              |
|    3           define pr_discos record like discos.*                     |
|    4                                                                     |
|    5           select * into pr_discos.*                                 |
|    6           from discos                                               |
|    7           where discos.numero = 1                                   |
|    8                                                                     |
|    9           display "DISCO 1:", pr_discos.nome                        |
|   10   end main                                                          |
|(discos.4gl:main)                                                         |
+--------------------------------------------------------------------------+
+--------------------------------------------------------------------------+
|$run                                                                      |
|Stopped in main at line 9 in module "discos.4gl"                          |
|$print pr_discos                                                          |
|discos.4gl:main.pr_discos  = record                                       |
|    numero = "1 "                                                         |
|    nome = "Plays Live          "                                        |
|    autor = "Peter Gabriel       "                                       |
|    faixas = "7 "                                                        |
|end record                                                                |
|$                                                                         |
+--------------------------------------------------------------------------+
```

Conforme se vê nas janelas, foi pedido o print da variável pr_discos, do tipo record.
Quando aparece o seu conteúdo, mais uma série de informação é especificada (nome do módulo, nome da função, tipo da variável).

8.2.Let

Atribui o resultado de uma expressão a uma variável

Tal como na própria linguagem 4GL a instrução *let* atribui a uma variável o resultado de uma função.
Pode, durante o exame a uma aplicação simular atribuições que deverão ser feitas.
A sua sintaxe é:

LET variável = expressão

As variáveis poderão ser quaisquer desde que definidas no programa e que sejam globais ou locais à função corrente.
De forma idêntica ao 4GL poderá também atribuir, no caso de strings, valores de partes de string(s) utilizando indexações.

A definição da variável obedece ao mesmo esquema definido para a instrução *print*.

```
+------------------------------------------------------------------------------+
¦    1   main                                                                  ¦
¦    2           define i smallint                                             ¦
¦    3           let i=25                                                      ¦
¦    4           display "hello world" at i,1                                  ¦
¦    5   end main                                                              ¦
¦                                                                              ¦
¦                                                                              ¦
¦                                                                              ¦
¦                                                                              ¦
¦(hello.4gl:main)                                                              ¦
+------------------------------------------------------------------------------+
+------------------------------------------------------------------------------+
¦(1)  break main:4 [hello.4gl]                                                 ¦
¦          scope function: main                                                ¦
¦$run                                                                          ¦
¦Stopped in main at line 4 in module "hello.4gl"                               ¦
¦$print i                                                                      ¦
¦hello.4gl:main.i  = 25                                                        ¦
¦$let i=23                                                                     ¦
¦$print i                                                                      ¦
¦hello.4gl:main.i  = 23                                                        ¦
¦$continue                                                                     ¦
+------------------------------------------------------------------------------+
```

Este exemplo mostra como uma variável que tem um valor indesejável (os écrans geralmente só têm 24 linhas) pode ser alterada, por forma a se poder continuar a examinar o programa sem ser confrontado com um erro fatal.

8.3.Dump

Mostra o conteúdo das variáveis.

Se tivermos um grande conjunto de variáveis, não se torna prático executar comandos print para cada variável, o que se poderia tornar muito extenso.

Para automatizar esta tarefa utiliza-se o comando *dump*. De seguida descreve-se genéricamente a utilização deste comando:

DUMP [GLOBALS | ALL] [>> NOME-FICHEIRO]

Este comando despeja as variáveis para o écran ou para dentro de um ficheiro (permite a utilização de redireccionamento).

As variáveis seleccionadas podem também ser seleccionadas pelo seu tipo. Assim se utilizar a opção GLOBALS só serão despejadas as variáveis globais.

Se escolher a opção ALL serão despejadas todas as variáveis (globais e locais à função corrente).

Caso não especifique nenhuma das opções serão seleccionadas apenas as variáveis locais à função corrente.

```
+--------------------------------------------------------------------------
--+
¦    1   main
¦
¦    2           define frase char(50)
¦
¦    3           let frase=arg_val(1)
¦
¦    4           display frase
¦
¦    5   end main
¦
¦
¦
¦
¦
¦
¦(hello.4gl:main)
¦
+--------------------------------------------------------------------------
--+
+--------------------------------------------------------------------------
--+
¦$break 4
¦
¦(1)  break main:4 [hello.4gl]
¦
¦          scope function: main
¦
¦$run "Hello World"
¦
¦Stopped in main at line 4 in module "hello.4gl"
¦
¦$dump
¦
¦DUMPING LOCAL VARIABLES OF FUNCTION [main]
¦
¦    frase = "Hello World                                       "
¦
¦$
¦
¦
+--------------------------------------------------------------------------
--+
```

Conforme experimentado no exemplo hello.4gl, fazendo o comando dump dentro de uma função, sem qualquer argumento, faz com que sejam despegadas as variáveis da função activa (isto é da função main).

8.4.Variable

Este comando serve para mostrar a linha de declaração de determinada variável ou variáveis.
Aliada ao redireccionamento poderá escrever o resultado num ficheiro.
A sua sintaxe genérica é:

VARIABLE [variável | GLOBALS | ALL] [>> nome-ficheiro]

Com o resultado deste comando virá também o nível scope (a função onde é reconhecida) a variável pedida.

Sem nenhum argumento for enviado ao comando, serão mostrados as declarações das variáveis locais à função correcta.

A especificação do nome da variável serve para visualizar apenas a declaração de uma variável, esse nome deverá referenciar uma variável global ou referente à função activa no momento.

Poderá, no entanto complementar o nome da variável com um qualificador de posição.

Esses qualificadores poderão ser:

GLOBAL - variáveis globais em todos os módulos

MODULE nome_módulo - variável global de determinado módulo

FUNCTION nome_função - Variável local a uma função

nome record [.nomerecord ...] - Variável pertencente a um record

Aproveitando o exemplo do capítulo anterior, vamos tentar ver qual o tipo da variável frase e ainda de algumas globais pré definidas.

```
+----------------------------------------------------------------------------+
¦    1   main                                                               ¦
¦    2           define frase char(50)                                      ¦
¦    3           let frase=arg_val(1)                                       ¦
¦    4           display frase                                              ¦
¦    5   end main                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
¦(hello.4gl:main)                                                           ¦
+----------------------------------------------------------------------------+
+----------------------------------------------------------------------------+
¦$variable frase                                                            ¦
¦hello.4gl:main.frase  type CHAR[50]                                        ¦
¦$variable global.status                                                    ¦
¦global.status  type INTEGER                                                ¦
¦$variable function.main.frase                                              ¦
¦function.main.frase  type CHAR[50]                                         ¦
¦$variable sqlca.sqlerrd                                                    ¦
¦global:sqlca.sqlerrd  type ARRAY [6] of INTEGER                            ¦
¦$                                                                          ¦
¦                                                                           ¦
+----------------------------------------------------------------------------+
```

Neste exemplo, aproveitou-se para verificar os tipos de um conjunto de variáveis, algumas locais, outras globais (pré-definidas).

Optou-se no entanto por efectuar esta tarefa utilizando sintaxes alternativas.

9.Outras Visualizações

9.1.Functions

O comando *functions* serve para ver as funções existentes no programa, ou para verificar se existe uma ou várias com o nome definido de determinada forma.
A sua sintaxe genérica é:

FUNCTION [pattern][>> nome_ficheiro]

Onde pattern poderá ser um conjunto de caracteres que simbolizem o nome da função. Nesta string poderá utilizar meta-caracteres.

9.2.List

Mostra breakpoints, tracepoints e configurações do écran

Para listar os pontos de paragem, pontos de tracejamento e configuraçãoda interacção dos écrans utiliza-se o comando *list*. Este comando poderá ter três argumentos, todos opcionais. Se nenhum deles fôr explicitado, será apresentado a informação como se tivesse digitado os três argumentos.

A sua sintaxe é:

LIST [BREAK] [TRACE] [DISPLAY]

Os parâmetros *break* e *trace* dizem respeito respectivamenteaos *breakpoints* e *tracepoints*. Estes são criados através dos comandos *break* e *trace.*

A opção display diz respeito à configuração do écran, sua interacção e respectivas janelas. Os parâmetros que são mostrados são os seguintes:

- AUTOTOGGLE - Define se há ou não mudança automática entre o écran da aplicação e do output.
- DISPLAY STOPS - Define se deve meter em reverse a própria lista a executar, quando o debugger pára.
- EXIT SOURCE - Especifica a forma como se passa de janela de source para janela de comando.
- PRINTDELAY - Atraso para mostrar uma linha ou várias linhas de uma vez.
- SOURCE LINES - Mostra o número de linhas que aparecem na linha de source.
- COMMAND LINES - Mostra quantas linhas aparecem na janela de comando.
- TIMEDELAY SOURCE -Define a pausa entre passos de execução quando *sourcetrace* está ligado.
- TIMEDELAY COMMAND - Define a pausa entre cada linha de comando que é mostrado na linha de comando.

- APPLICATION DEVICE - Mostra, caso exista, o nome terminal especificado para mostrar a aplicação.

Para alterar estes dados utiliza-se o comando TURN, GROW, etc.

9.3.Where

Apresenta a pilha de funções e respectivos argumentos de execução.
Quando se analisa um programa, muitas vezes necessitamos saber quais as funções activas em determinado momento e quais os argumentos com que foram executados.
Com a instrução *where* é apresentada essa informação, e este pode ainda ser apresentado para dentro de um ficheiro (utilizando o redireccionamento).

```
WHERE  [>> nome-ficheiro]
```

Esta instrução é ainda válida se o programa abortar com um erro fatal. Esta capacidade é muito importante uma vez que permite saber as funções activas (caminho de execução) na altura em que o programa deu erro.
Não pode utilizar esta instrução se não estiver nenhum programa ou função activa.

10.Paragens de Programas

10.1.Pontos de Paragem

Uma das características fundamentais de qualquer debugger é o ponto de paragem.
Um ponto de paragem é um ponto específico do programa, onde se define que o programa deve parar a execução (mediante ou não determinadas condições) e devolver o controlo para a janela de comando.

O informix ID é particularmente completo nesta característica.

10.2. BREAK

Para estabelecer um ponto de paragem utiliza-se o comando BREAK. este comando tem uma sintaxe complexa, uma vez que permite uma grande diversidade de especificação de paragens, nomeadamente:

- Por função
- Por linha de módulo
- Condicional ao conteúdo de variáveis
- Execução de comandos após paragem
- Parar ao fim de uma quantidade fixa de passagens por esse ponto

A sintaxe do BREAK é:

```
BREAK [*] [(função)] ["nome-break-point"] [-contador]

 {[[módulo.] no-de-linha | variável | função] [IF condição]}

[{comandos [ ;comandos...]}]
```

Tal como pode verificar, a sua sintaxe é de facto complicada. Por forma a explicar mais fácilmente este comando vamos dividir por tipos breakpoints mostrando a sintaxe envolvida e dando um máximo de exemplos.

Antes de passarmos a explicações e casos específicos vamos esclarecer algumas definições:

Nome do breakpoint

A um *breakpoint* pode ser dado (opcionalmente) um nome específico. Este não deverá ser referenciado entre aspas e será por ele que um *breakpoint* passará a ser conhecido.

Número de referência de um breakpoint

Além do nome, um breakpoint pode ser referenciado pelo seu número de referência, que é sequêncial por ordem de estabelecimento de *breakpoint.* Nesta marcação entram também os tracepoints.

Enable e disable de breakpoints

Enable - Activar

Embora definido, por vezes não se pretende que o ponto de paragem esteja ainda activo, por forma a conseguir simular a situação pretendida. A acção de activar um breakpoint é feita com o comando *enable* cuja sintaxe de seguida se descreve:

ENABLE {nome | número- de-refeência | ALL}

Conforme se pode observar, pode-se activar *breakpoints* pelo seu número de referência ou pelo seu nome. Pode ainda activar todos os *breakpoints* inactivos.

O nome a especificar poderá ser um dos seguintes:

- nome do *breakpoint* se este estiver entre aspas.
- nome do *breakpoint* ou *função* se este não estiver entre aspas.

Disable - desactivar

Um breakpoint pode ser desactivado. Esta acção não remove o breakpoint, apenas o adormece até que seja novamente activado. Quando a execução do programa passa por uma zona onde haja um breakpoint desactivado, a sua execução obviamente que não pára.

A sua sintaxe é a seguinte:

DISABLE { nome | número-de-referência | ALL}

Tal como no enable pode referênciar um *breakpoint* específico (por nome ou número de referência) ou todos (utilizando a palavra ALL). Um *breakpoint* pode ser criado já desactivado, se quando o criar especificar a opção "*", imediatamente a seguir à palavra BREAK.

10.2.1.Definição do ponto específico

O ponto de paragem diz, em geral sempre referência a uma linha específica de um módulo específico do programa a examinar.

Pode-se especificar este ponto através da sua definição exacta (módulo e linha) utilizando:

modulo-nº-de-linha

O ponto de paragem pode também ser definido através do ponto em que o conteúdo de variáveis fôr alterada, utilizando a seguinte sintaxe:

BREAK variável

Pode-se ainda definir um breakpoint através do nome da sua função. Se tal acontecer, a execução do programa será interrompida assim que se entrar na função referida.

A sua sintaxe reduzida é:

BREAK função

10.2.2.Paragem Simples Incondicional

Este tipo de paragem é muito utilizada sempre que se pretenda parar num ponto específico, e conhecido do programa, para depois analisar o ambiente com os outros comandos.

A sua sintaxe reduzida é:

BREAK [*] "nome-break" definição-ponto

Nesta paragem o programa para a sua execução assim que passar naquele ponto, independentemente de quaisquer valores de variáveis. A *definição ponto*, tal como já foi explicado poderá ser:

Módulo. nº-linha - Se não especificar o módulo será assumido o corrente.

Variável - Pára quando a variável especificada fôr alterada.

Função - Pára assim que o programa entrar na função especificada.

10.3.Paragem Condicional

Muitas vezes, durante uma sessão de exame a um programa, só se pretende parar a execução caso determinada situação ocorra (por exemplo determinada variável conter um valor negativo). Para detectar estes casos utiliza-se a paragem condicional, que não é mais que dizer ao debugger para *parar se determinada situação ocorrer*.

A sua sintaxe genérica é:

BREAK identificação-break IF condições

A *identificação-break* representa todos os parâmetros que nos permitem definir um breakpoint (disable, nome, ponto de paragem), ou poderá não ser nenhum deles uma vez que podemos querer parar se determinada situação ocorrer, independentemente do ponto onde nos encontrarmos.

A condição explicitada a seguir ao IF, mais não é que uma condição normal em informix 4GL onde poderão ser utilizadas variáveis (desde que activas), constantes, etc. A definição destas variáveis poderá ser feita de acordo com o definido para o comando *print.*

10.4.Paragem com execução de comandos

Para executar determinados comandos imediatamente a seguir a uma paragem (p/ ex. o print), situaçãoque ocorre muitas vezes poderá utilizar a capacidade do comando break que o permite fazer. A sua sintaxe reduzida será:

BREAK identificação-break [IF condição]

{comandos [; comandos ...]}

Conforme pode ser verificado poderá meter vários comandos desde que separados por ";". As chavetas são obrigatórias, não tendo qualquer significado especial de definição de sintaxe.
Estes breakpoints podem ser óbviamente condicionais em incondicionais consoante especificar ou não o comando IF.

10.5.Paragem depois de uma quantidade de passagens

Por fim podemos ainda parar em determinado sítio, depois de lá passar um número de vezes por nós especificado (por exemplo se tivermos um ciclo FOR e só nos interessar a última interacção).
A sua sintaxe é a seguinte:

BREAK definição-break -count

condições-e-execuções

Consoante o número específicado no lugar de *count*, assim será o número de vezes que o programa passará naquele ponto sem efectuar qualquer paragem.
De notar que caso tenha um IF, o contador só será decrementado caso a expressão dê um valor verdadeiro.

10.6.Desactivação, activação e remoção de pontos de paragem

Para activar e desactivar breakpoints, tal como já atrás foi descrito utilizam-se os comandos *enable* e *disable*.Para remover **efectivamente** os pontos de paragem
utiliza-se o comando *nobreak.*

A sua sintaxe é a seguinte:

NOBREAK {nome | nº_referência | função | ALL}

Esta sintaxe funciona da mesma forma que para o *enable* e *disable*. Assim, podemos remover todos os breakpoints se utilizarmos o argumento *all*, ou remover um breakpoint especifico identificando-o de uma das seguintes formas:

nome - se entre aspas é obrigatóriamente um nome, senão arrisca-se a remover um breakpoint de uma função

nº_referência - o seu número sequêncial de criação

função - não pode pôr entre aspas, basta digitar o nome da função

Mais uma vez se repete que o *nobreak* remove definitavamente um breakpoint, enquanto que o *disable* apenas o desactiva.

10.7.Listar breakpoints

Se pretender ver a definição dos breakpoints pode utilizar o comando ***list*** com a opção break. A sua sintaxe é a seguinte:

LIST BREAK

11.Tracejamento

O tracejamento serve para que o debugger assinale quando passa em determinado parte do programa ou quando determinada variável é alterada e para que valor.

Se não for especificada qualquer redireccionamento, o resultado será enviado para a janela de comando. Se pretender analisar posteriormente o resultado, poderá enviar o resultado para um ficheiro.
A sua sintaxe genérica é a seguinte:

TRACE [*] [(função)] ["nome"]

{[módulo.] remova-de-linha | variável | função | FUNCTIONS}

[{comandos}] [>> nome-de-ficheiro]

O funcionamento desta capacidade é muito idêntica aos breakpoints, relativamente à definição do ponto de paragem, podendo ser:

[módulo.] número-de-linha -	Mostra a linha especificada do módulo especificado. se não especificar o módulo pára no módulo corrente.
variável -	Sempre que o conteúdo da variável cujo nome se específica fôr alterado, é enviada a informação de tracejamento.
Função -	Mete um ponto de tracejamento na função cujo nome se especifica.
FUNCTIONS -	passa a existir um tracepoint no ínicio de cada função.

Tal como nos breakpoints, um tracepoint pode ser criado sem estar activo, utilizando para tal o caracter "*" a seguir ao comando *trace* .

A um tracepoint é sempre atribuído um número sequêncial de referência. Este número é sequêncial nos *break(s)* e *trace(s)* fazendo com que não haja um *breakpoint* e *tracepoint* com o mesmo número de referência.

De seguida vai-se tentar dar uma retrospectiva dos tipos de *tracepoints* mais utilizados.

11.1.Tracepoint - a passagem por função

Neste tipo de tracepoint apenas se identifica o tracepoint relativamente à função. A sua sintaxe reduzida é:

TRACE [*] ["NOME"]

FUNÇÃO [>> nome-do-ficheiro]

11.2.Tracepoint e alteração de variável

Tal como no caso anterior basta identificar a função à qual se pretende fazer breakpoint.

TRACE [*] ["nome"] variável [>> nome-ficheiro]

O nome da variável obedece à sintaxe utilizado no IF dos breakpoints e no print. Isto é:

11.3.Tracejamento de variável apenas numa função

Poderá querer apenas ver a alteração que determinada variável sofra, mas apenas dentro de uma variável. Para tal deverá especificar o nome da função entre parêntesis.

A sua sintaxe poderá ser descrita da seguinte forma:

TRACE [*] (função_pretendida) variável [>> nome_ficheiro]

11.4.Tracepoints executando comandos

Tal como nos breakpoints, também aqui poderá executar determinados comandos quando passar num ponto de tracejamento.

TRACE [*] definição_do _tracepoint

{comandos} [>> nome_ficheiro]

Os comandos também aqui podem ser um conjunto desde que separados por ponto e virgula(";") e desde que não sejam nenhum dos seguintes: CALL; CONTINUE; STEP; RUN. As chavetas servem para delimitar o início e fim dos comandos a executar.

11.5.Activação de tracepoints

Para activar um tracepoint utiliza-se o comando *enable*. Uma vez que os breakpoints e tracepoints são identificados de uma forma univoca (isto é, não existem breakpoints e tracepoints com uma referência e nomes iguais) o comando enable serve para as duas coisas. Óbviamente que para activar um tracepoint, este deve existir e estar desabilitado.

ENABLE { nome | nº_referência | função |ALL}

11.6.Desactivação de tracepoints

Um tracepoint pode ser desactivado utilizando o comando DISABLE, tal como nos breakpoints ou criando o tracepoint já desactivado utilizando o caracter "*".

DISABLE { nome | nº_referência | função | ALL}

11.7.Remoção de tracepoints

A remoção de um ponto de tracejamento é feito com o comando NOTRACE. A sintaxe genérica é a seguinte:

NOTRACE { nome | nº_referência | função | ALL}

A identificação do tracepoint pode ser feita atrvés do seu nome: número de referência; função; ou desactivar todas os tracepoints existentes.

12.Escrita e leitura de uma sessão de debugg

Uma das facilidades existentes para aumentar a produtividade do debugg é a possibilidade de escrver e ler (parcialmente ou não) todo o ambiente das sessões de debugg. Para além disso os ficheiros utilizados são estritamente de texto, por conseguinte poderão eventualmente ser criados atrvés do editor de texto.

12.1.Escrever (guardar) sessão de debugg

12.1.1.Write

Mais atrás foi referida que a sessão poderia ser guardada e ainda são parcialmente. Para tal utiliza-se o comando *write* que se utiliza da forma que de seguida se descreve:

WRITE [BREAK] [TRACE] [DISPLAY] [ALIASES] [>>] [nome_ficheiro]

Conforme se pode perceber o comando é composto por quatro partes: O seu nome própriamente dito; os parâmetros a salvaguardar; as opções de escrita e o nome do ficheiro.

Os parâmetros a salvaguardar poderá ser qualquer dos seguintes e poderão ser utilizados vários de cada vez:

BREAK: Guarda os breakpoints definidos
TRACE: Guarda os tracepoints definidos
ALIASES: Guarda todos os aliases existentes
DISPLAY: Escreve a configuração dos parâmetros da

Se reparar o redirecionamento (>>) é opcional. Se o incluir nos parâmetros a guardar serão escritos no final do ficheiro. Se não incluir o ficheiro será limpo e começando a escrever do ínicio.

O nome do ficheiro poderá ou não ser definido com a extensão *.4db, no* entanto ficará sempre guardado com a respectiva extensão.

Se não especificar o nome do ficheiro onde pretende guardar os parâmetros (uma vez que é opcional) estas serão escritas num ficheiro com o nome do programa que estamos a examinar, substituindo a sua extensão por *.4db*.

12.1.2.Read

Lê comandos de um ficheiro de texto.

Pode-se escrever uma série de comandos do debugger num ficheiro de texto. Se forem comandos de configuração basta utilizar o comando *write.* Se forem outros comandos utiliza-se um qualquer editor de texto.

Para ler do ficheiro executar estes comandos utiliza-se o comando *read* cuja sintaxe de seguida se descreve:

READ nome-do-ficheiro [.4db]

O nome do ficheiro tem obrigatóriamente de ser especificado, sendo opcional a discriminação da extensão *.4db*. Se esta não fôr especificada é automáticamente assumida.

O ficheiro a ler poderá ainda ter um comando *read,* no entanto não poderá passar um máximo de 10 níveis de *read(s).*

13.Casos concretos de análise

13.1.Analisar erros fatais.

Este é um caso tipico de exame que deve ser feito a um programa, uma vez que acontece com relativa frequência e é fácil de reproduzir.

Os passos tipicos a efectuar uma sessão de debug são:

- Reproduzir o erro.
- Descobrir módulo e linha onde aborta (logfile ou mensagem de erro).
- Alterar caminhos de busca para permitir encontrar módulos envolvidos no erro (instrução use).
- Verificar e analisar todo o ambiente existente na altura do erro (run, where, dump).
- Criar conforme necessário pontos de paragem e tracejamento a variáveis ou funções para análise.
- Se o responsável fôr o conteúdo de uma variável deverá efectuar o teste, fazendo uma atribuição com *let,* de um valor correcto.

13.1.1.Uma variável não tem os volumes que deveria ter.

Se detectou que uma variável não contém um valor corrente, isto é um valor que permita um correcto funcionamento do programa a receita para efectuaros testes necessários é:

- Fazer uma frase a essa variável.
- Descobrir onde a variável é afectada com um valor incorrecto.
- Simular uma execução efectuada depois deste ponto a variável com um valor correcto
- Se o programa funcionar deverá alterar de forma definitiva no source e voltar a compilar.

13.1.1.1.O fluxo do programa não segue o que se pretende

- Testar a funcionalidade de uma função.
- O cliente identificou um erro de lógica pelo écran, e uma vez que não fomos nós a fazê-lo e pretendemos descobrir em que função e módulo aparece o erro.
- As variáveis estão a ficar todas baralhadas.
- Cada vez que querem ver uma variável tenho de escrver um comando muito extenso.

- Tenho de ir almoçar e quero guardar os meus brakpoints, tracepoints e aliases.
- Um array foi utilizado fora da sua dimensão.

Glossário

Argumento - Opção que se passa a comandos, funções ou programas que lhes permitem variar a forma de execução.

Redireccionamento - Mandar o output de um comando na direcção de um ficheiro.

Breakpoint - Ponto de paragem da aplicação.

Tracepoint - Ponto onde será tracejado que o programa lá passou e eventualmente alterou uma variável.

ÍNDICE REMISSIVO